Problem Set 2 Solution

AA274: Principles of Robotic Autonomy
Stanford University
Winter 2018

Chi Zhang
SUNet ID: czhang94
Date: February 18, 2018

Problem 1: Camera Calibration

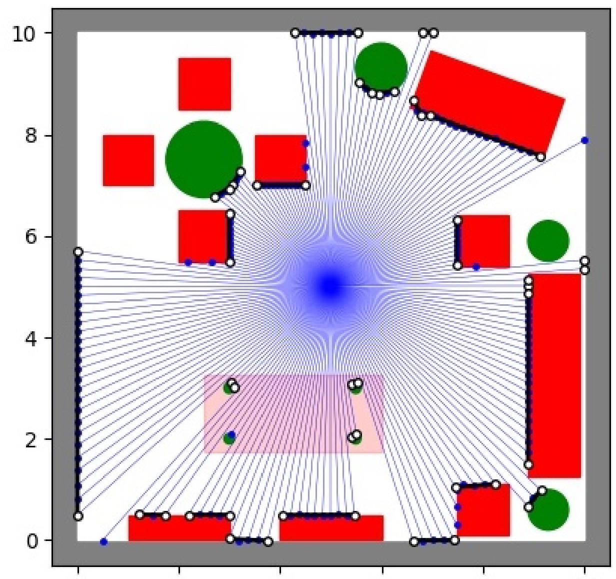
No writeup is required. Please see the code in `cam_calibrator.py`.

Problem 2: Line Extraction

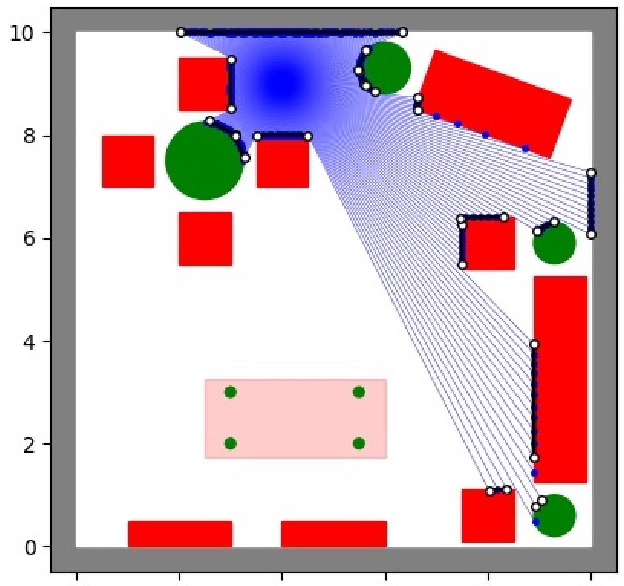
The segmentation parameters are

Parameters	Values
`LINE_POINT_DIST_THRESHOLD`	0.03
`MIN_POINTS_PER_SEGMENT`	2.0
`MIN_SEG_LENGTH`	0.01
`MAX_P2P_DIST`	0.23

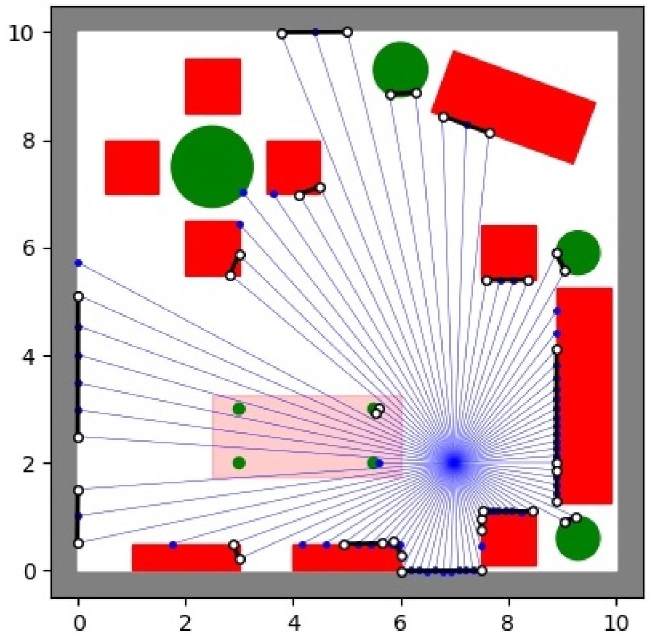
The three plots are shown below:



(a) `rangeData_5_5_180.csv`



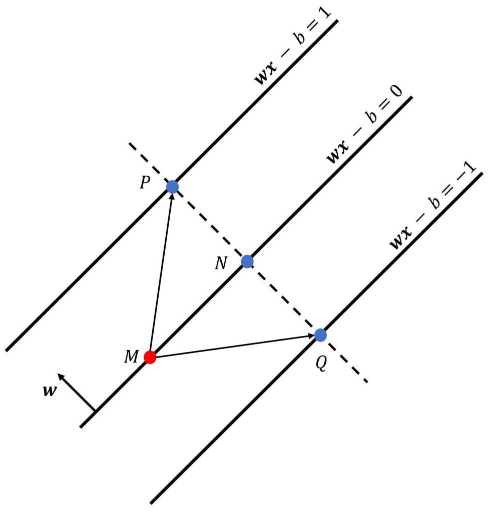
(b) `rangeData_4_9_360.csv`



(c) `rangeData_7_2_90.csv`

Problem 3: Tensorflow and HOG+SVM Pedestrian Detection

(i) *Proof.* As shown in the figure below, the perpendicular distance between the two planes is $PQ = PN + NQ$.


$\boldsymbol{wx} - b = 1$
$\boldsymbol{wx} - b = 0$
$\boldsymbol{wx} - b = -1$
P
N
M
Q
$\boldsymbol{w}$

Denote two vectors as

$$\overrightarrow{MP} = \begin{bmatrix} x_{p1} - x_{m1} \\ x_{p2} - x_{m2} \end{bmatrix} \quad \overrightarrow{MQ} = \begin{bmatrix} x_{q1} - x_{m1} \\ x_{q2} - x_{m2} \end{bmatrix}$$

Therefore,

$$PN = \left| \frac{\vec{w} \cdot \overrightarrow{MP}}{\|\vec{w}\|} \right| = \left| \frac{\vec{w} \begin{bmatrix} x_{p1} \\ x_{p2} \end{bmatrix} - \vec{w} \begin{bmatrix} x_{m1} \\ x_{m2} \end{bmatrix}}{\|\vec{w}\|} \right| = \left| \frac{1-0}{\|\vec{w}\|} \right| = \frac{1}{\|\vec{w}\|}$$

$$NQ = \left| \frac{\vec{w} \cdot \overrightarrow{MQ}}{\|\vec{w}\|} \right| = \left| \frac{\vec{w} \begin{bmatrix} x_{q1} \\ x_{q2} \end{bmatrix} - \vec{w} \begin{bmatrix} x_{m1} \\ x_{m2} \end{bmatrix}}{\|\vec{w}\|} \right| = \left| \frac{-1-0}{\|\vec{w}\|} \right| = \frac{1}{\|\vec{w}\|}$$

$$PQ = PN + NQ = \frac{2}{\|\vec{w}\|}$$

□

(ii) Tensorflow computational graph is an alternative way of conceptualising mathematical calculations. Consider the following operation: $c = (a + 2) + (a \times b)$. The operation can be broken down into the following components:

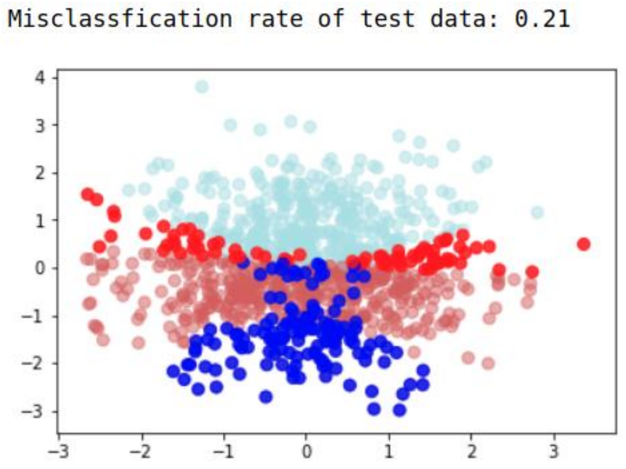
$$d = a + 2$$

$$e = a * b$$

$$c = d + e$$

Note that two of the computations $d = a + 2$ and $e = a \times b$ can be performed in parallel. By splitting up these calculations across CPUs or GPUs, this can give us significant gains, which cannot be obtained using numpy, in computational times. Another advantage of computational graph is its automatic derivative computation, whereas you have to implement a lot of functions to compute all derivatives.

(iv) The misclassification rate of the test dataset is 0.21. With identity features, the classifier can only learn a separation line in the middle of data plane, which results in misclassifying nearly all blue data points below the line.

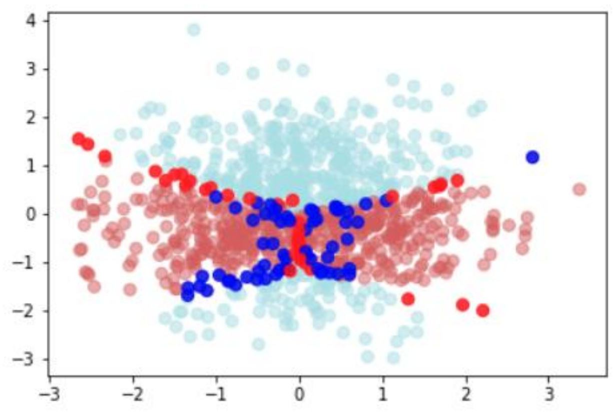


(vi) The features I chose are x_1, x_2, $e^{|x_1|}$, $e^{|x_2|}$, $\left|\frac{x_2}{x_1}\right|$ and $|x_1||x_2|$. Here are reasons why I chose these features besides identical features:

- Exponential features $e^{|x|}$ can magnify the difference between a small and a big $|x|$, working as a fancy version of powers of x. It is a good choice in that one class of data has relatively large $|x_1|$ while the other $|x_2|$.
- A common choice for machine learning in general is to choose monomials as features. In 2D cases, it is wise to include cross term $|x_1||x_2|$ in addition to powers of x_1 and x_2.
- Another fancier feature is the absolute value of the slope of line segment from the origin to a data point $\left|\frac{x_2}{x_1}\right|$. It is chosen because it perfectly describes geometric distribution of two classes of data.

Please note that the absolute value takes care of symmetric distribution of data. Using custom features, the misclassification rate of test data is reduced to 0.097, as shown below.

```
Misclassfication rate of test data: 0.097
```

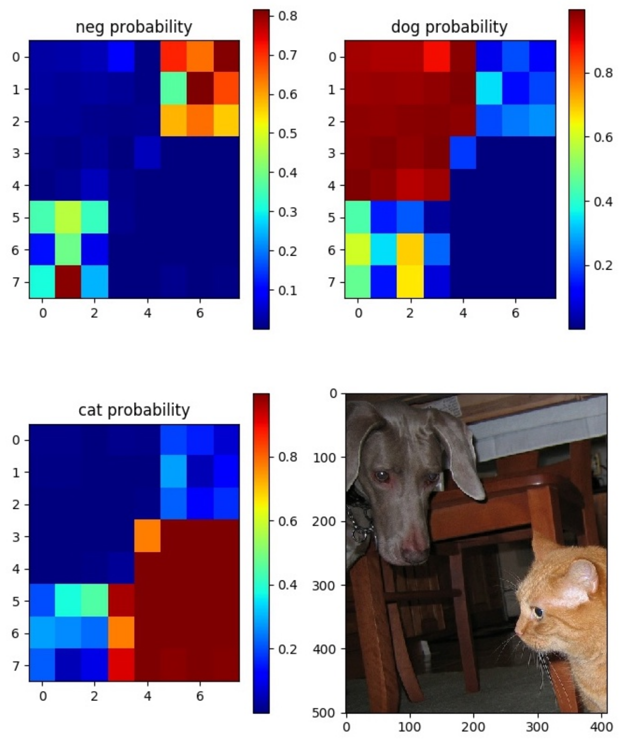


(viii) The misclassification rate on the test set is 0.045.

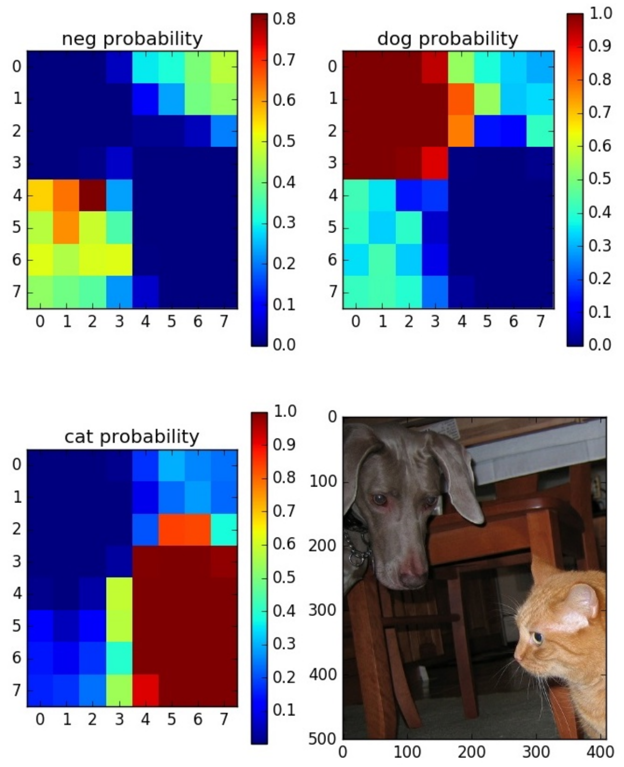
Problem 4: Classification and Sliding Window Detection

(i) The "bottleneck" image summary is a 2048-dimensional vector. For N images, the summary becomes a $N \times 2048$ matrix. The size of weights matrix and bias vector are 2048×3 and 3×1 respectively, thus there are $(2048 \times 3 + 3 =)$ 6147 parameters optimized in the retraining phase.

(iii) The detection plot generated by brute force sliding window method is

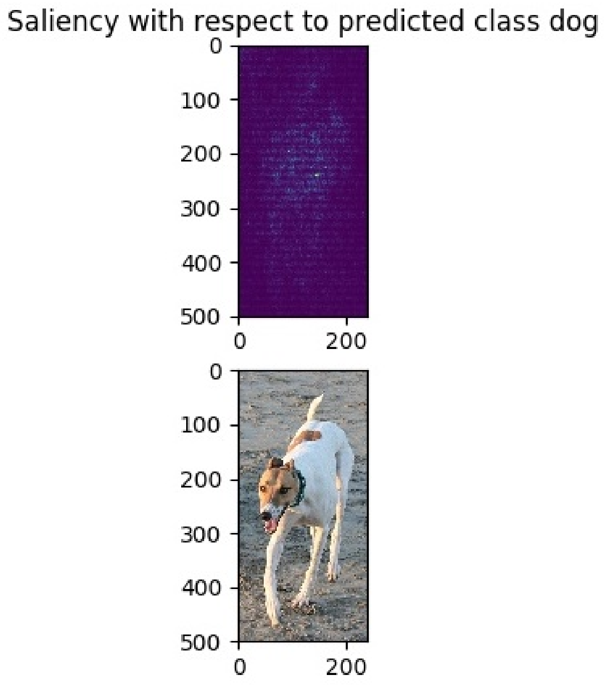


(iv) The detection plot generated using already computed image features is

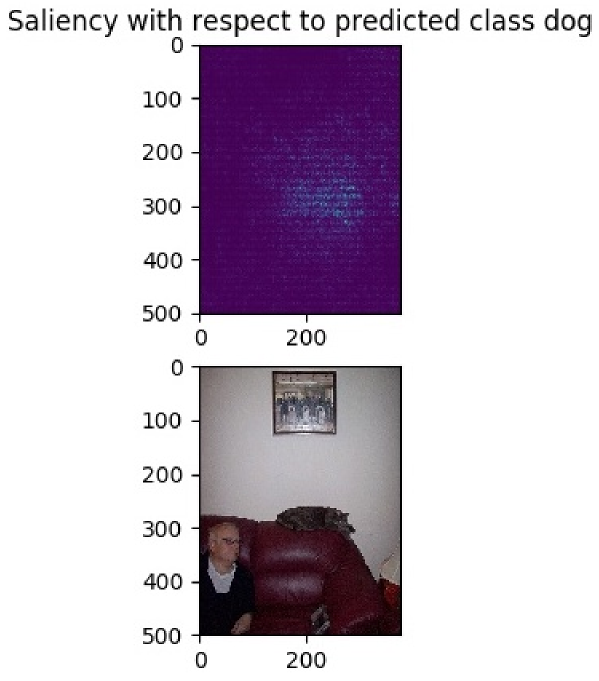


(v) It is fed into average pooling. The feature vector for the whole image is computed as average of all sub-region feature vectors.

(vii) The saliency value for a pixel represents how much it contributes to classifying the image as class c. For a correctly classified image, "highlighted" pixels are typically the most distinguishable features of the class, such as pixels corresponding to the dog's head, body and part of limbs in figure (a). For an incorrectly classified image, "highlighted" pixels are the pixels to blame for misclassification. In figure (b), the cat pixels lead to misclassification as dog though it is really tricky to distinguish the two furry animals in this case.



(a) correctly classified as dog



(b) incorrectly classified cat as dog

Figure 2: saliency map of both a correctly and incorrectly classified image

Problem 5: Stop Sign Detection and FSM in ROS

(i) It publishes the current desired pose to the pose controller. The message type is `Pose2D`.

(vi) The state diagram is shown in Fig.3.

(viii) The path and velocity profile is shown in Fig.4.

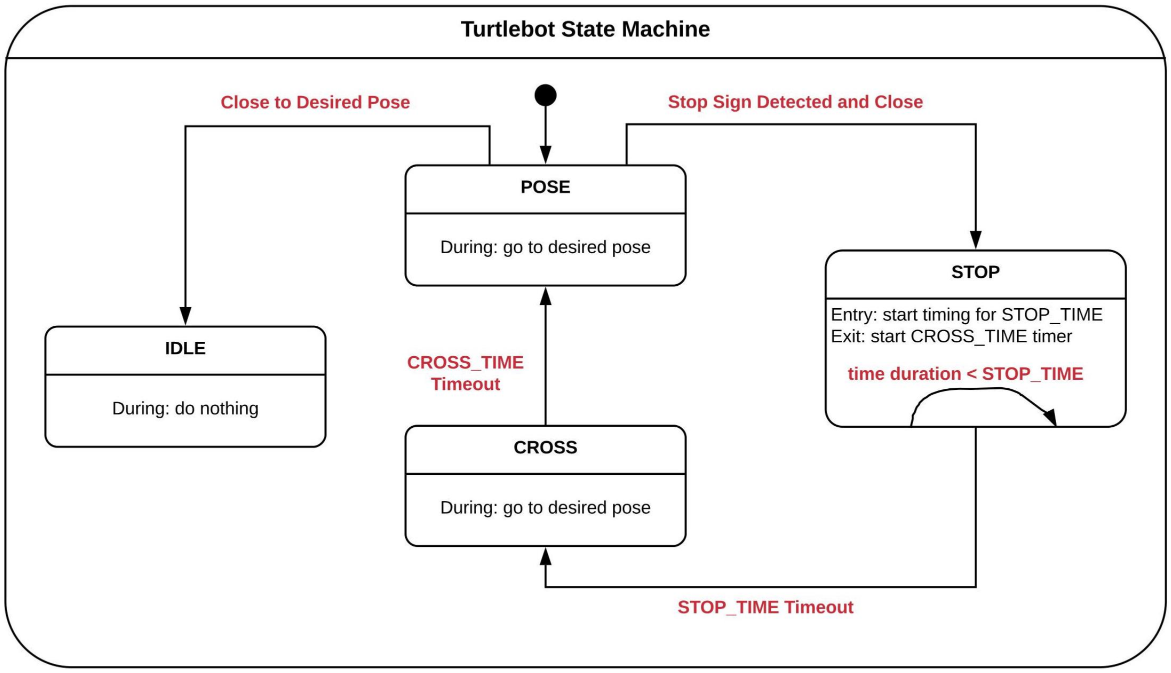

Turtlebot State Machine
Close to Desired Pose
Stop Sign Detected and Close
POSE
During: go to desired pose
STOP
Entry: start timing for STOP_TIME
Exit: start CROSS_TIME timer
time duration < STOP_TIME
IDLE
During: do nothing
CROSS_TIME Timeout
CROSS
During: go to desired pose
STOP_TIME Timeout

Figure 3: state diagram of FSM

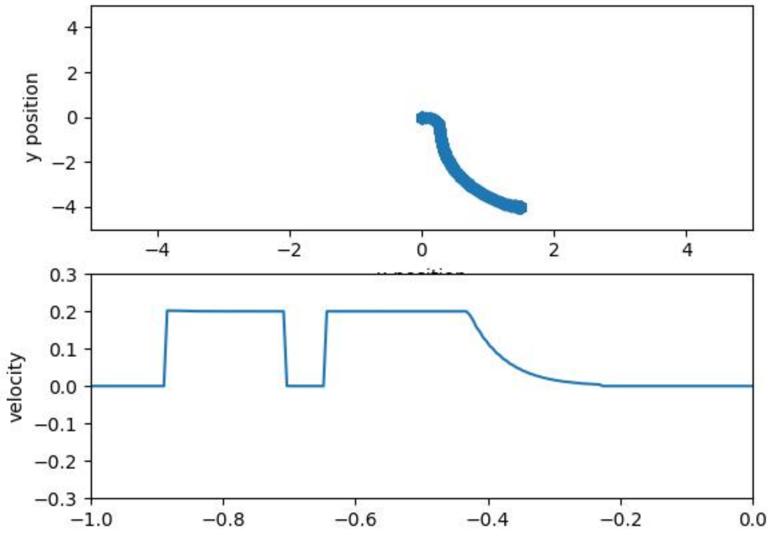


Figure 4: path and velocity profile